Self-intro:

Name + years + current project(what company) + stack (backend + databases + frontend + tools) orm tools as well

Hello, my name is XXX, and I have been working as a full stack .NET developer for approximately 5 years, and I am currently working on a project at Lyra Health. For backend development, the technologies that I'm familiar with are C#, ASP.NET/ASP.NET Core, EF/EF Core, and ADO.NET. For databases, I am experienced in relational databases such as MS SQL Server. For Frontend,I am familiar with Angular 2+, and Razor Views. (**other production tools need to add:**) Git for version control, Agile for project management, xUnit and Moq for testing, and RabbitMQ for messaging. So this is pretty much about myself, is there anything you would like me to elaborate on?

Summary intro about the company:

What is the name of the company, main business, what is the project doing, what are the functionalities that I worked on, what was the tech used.

The company is called Lyra Health, and it is essentially a company that provides convenient help for people who are struggling with their mental health and are actively seeking help.

So basically, my team and I were responsible for developing and maintaining the application overall, and I was involved in developing provider matching functionality, for that, I participated in the development of the survey module, graduate module, and provider module. This is a restful web application, and some of the tech stacks we used are: C#, ASP.NET Core, Entity Framework Core, SQL Server, xUnit, Moq, Azure DevOps, Angular2+, Git

(latest one)

The client would fill out that survey which included information such as their location, availability, type of help they need, language, and additional things needed to know by the provider. The survey results would then be sent from the survey module to a composite module. The results would then be sent to both the provider and graduate module. The graduate module contains clients who have "graduated" from Lyra Health. Each graduate contains information such as their provider, the type of help they need, a review of their provider, and the score given to the provider. The composite will then retrieve the graduates with the same type of mental health issue. Based on that information, the provider module will then perform some business logic in order to find providers that have good average scores while also matching with the client information given to the survey. As the database grows, it will take longer to retrieve the information for the client. As a result, we added RabbitMQ to send the information to the client through an email module.

I guess the business logic would be finding the providers with the highest average ratings.

Challenge was that we were trying to find the right solution without touching our code: - With database indexing, we wouldn't really have to modify our code that much - You don't have to do as much modification with async, but that didn't really work either since the services were not independent

- In the end, we figured the best course of action was to use RabbitMQ, even though we'd have to modify our code quite a bit.

One detailed example of your workflow:

The client would fill out that survey which included information such as their location, their availability, the type of help they need, other personal information, and additional things needed to know by the provider. The survey results would then be sent from the survey module to the provider module using httpClient. Based on the survey, the provider module would give a list of providers from its own independent database that would best suit the client based on the results given through the survey. That list would then be sent back to the clients using RabbitMQ (temporary). The list would then be sent back to the recommendation module and sent to the view.

Lyra Health story:

So basically, my team and I were mainly responsible for the provider matching functionality and maintaining the application overall. So I was involved in helping out in developing this functionality. I participated in the development of the registration module, appointment module, and provider module. My team started out trying to add extra features to the registration module such as making the frontend look more visually appealing (so basically the style of the frontend). Since I did not really have experience in this area, this task was assigned to other people. Our team was deciding on what type of architecture to use between monolithic and microservice and ultimately decided on the microservice architecture. For the provider matching functionality, we had a recommendation module that acted as a survey for the client. The client would fill out that survey which included information such as their location, availability, type of help they need, language, and additional things needed to know by the provider. The survey results would then be sent from the survey module to the provider module by using HttpClient. Based on the survey, the provider module would give a list of providers from its own independent database that would best suit the client based on the results given through the survey. That list would then be sent back to the clients using RabbitMQ (temporary). The list would then be sent back to the recommendation module and sent to the view. In order to test our services and controllers step by step, we used Moq and xUnit..

-

The graduate module contains the providers who have clients that have “graduated” from Lyra Health (which means they now no longer need Lyra Health to help with their issues). Each provider in the graduate database contains the type of mental health issue they specialize in, the number of people that they have helped graduate, a list of reviews from the clients, and an overall score.

The survey results would then be sent from the survey module to the provider module by using HttpClient. Based on the survey, the provider module would give a list of providers from its own independent database that would best suit the client based on the results given through the survey. That list would then be sent back to the clients using RabbitMQ (temporary). The list would then be sent back to the recommendation module and sent to the view

Email module listening to rabbitmq. Once a new message is pushed into the queue, it will send an email based on the message.

- Message has email address, content of email, etc

We tried a bunch of other solutions such as DB indexes, async but they didn't work because tasks were not independent.

- Tried a lot of things to identify the best solution which was pretty time consuming. - It was hard to figure out which message queue to use exactly
- Come up with the reason why you chose rabbitmq over kafka
    - Initially we were figuring out the differences between RabbitMQ and Apache Kafka.
    - Ultimately we decided to use RabbitMQ since we thought the idea of a messaging queue fit better with our application rather than apache kafka's pub sub method.
    - With RabbitMQ, it was easier to send certain messages to specific consumers as opposed to Kafka.
        - Each consumer has its own consumer queue and certain messages will be sent to that specific queue.
        - Producer publishes a message along with a routing key and matches the routing key to the binding key of the consumer queues.

Another one issue that arose was performance. Since it took some time for the provider module to give back the client the list of suitable providers ( most of the time it would be fairly quick but if there was information filled in the additional things then it would take some time for the client to receive the results), we decided to use RabbitMQ for offline transactions. After we were able to decide which providers are suitable, we would send the list of clients as a message back to the clients.

Challenge was that we were trying to find the right solution without touching our code: - With database indexing, we wouldn't really have to modify our code that much - You don't have to do as much modification with async, but that didn't really work either since the services were not independent

- In the end, we figured the best course of action was to use RabbitMQ, even though we'd have to modify our code quite a bit.
- We didn't really want to modify our code in the beginning since we did not want any unnecessary bugs that would come along the way after working on our code for so long.

Ginger Story:

Basically, my team and I were responsible for the user profile, the matching caretaker functionality, and the recommended articles functionality while also maintaining the application in general (making sure everything works fine. We tried to make sure that if one service goes down it wouldn't affect the others). I was mainly involved with backend development and parts of the frontend. I was not involved with the style of the frontend since I was not experienced enough in that area, and we had team members who were quite exceptional at it. I participated in the development of the user profile service, the caretaker service, and the recommended article service. Our team initially was deciding one what type of architecture to use and ultimately decided on microservice architecture since we preferred having independent applications. We also thought that if we wanted to include another service that used different technologies then using a microservice architecture would be much more beneficial. For the recommended article service, a client would select topics that interested them. Based on a variety of topics, the client would get back some recommended articles that were related to those topics. After the client submits the list of topics, the topics module would get sent to the recommended articles module by using the httpclient. The list of recommended articles would then be displayed to the view and the client would be able to scroll through the different articles. However, the application would run into quite some problems during my working time. Since at this time was when COVID-19 first hit and many places closed down, a lot of people started to struggle with their mental health because they were unable to communicate with their loved ones and also could not go to places they wanted to go to. So as users of the Ginger application increased, the many requests that came in would ultimately lead to cascading failure. This was one of the challenges we faced since as many requests would be made to certain services which ultimately lead to the failures of others. In order to solve this issue, we utilized Resilience 4J circuit breaker in order to prevent these cascading failures.

Basically we have a survey service, composite service, caretaker service and recommended articles service. When first entering Ginger, the client would fill out information in the survey and that survey would be sent to a composite service. The composite service would then grab information from both recommended articles service and caretakers service. Since both of them were independent services, we figured that we could use async to grab both of them at the same time in order to improve the performance of our application.